Dynamic and Static SQL Support

This section covers the following topics:

- General Information
- Internal Handling of Dynamic Statements
- Preparing Natural Programs for Static Execution
- Assembler/Natural Cross-References
- Execution of Natural in Static Mode
- Static SQL with Natural Security
- Mixed Dynamic/Static Mode
- Messages and Codes
- Application Plan Switching in Static SQL

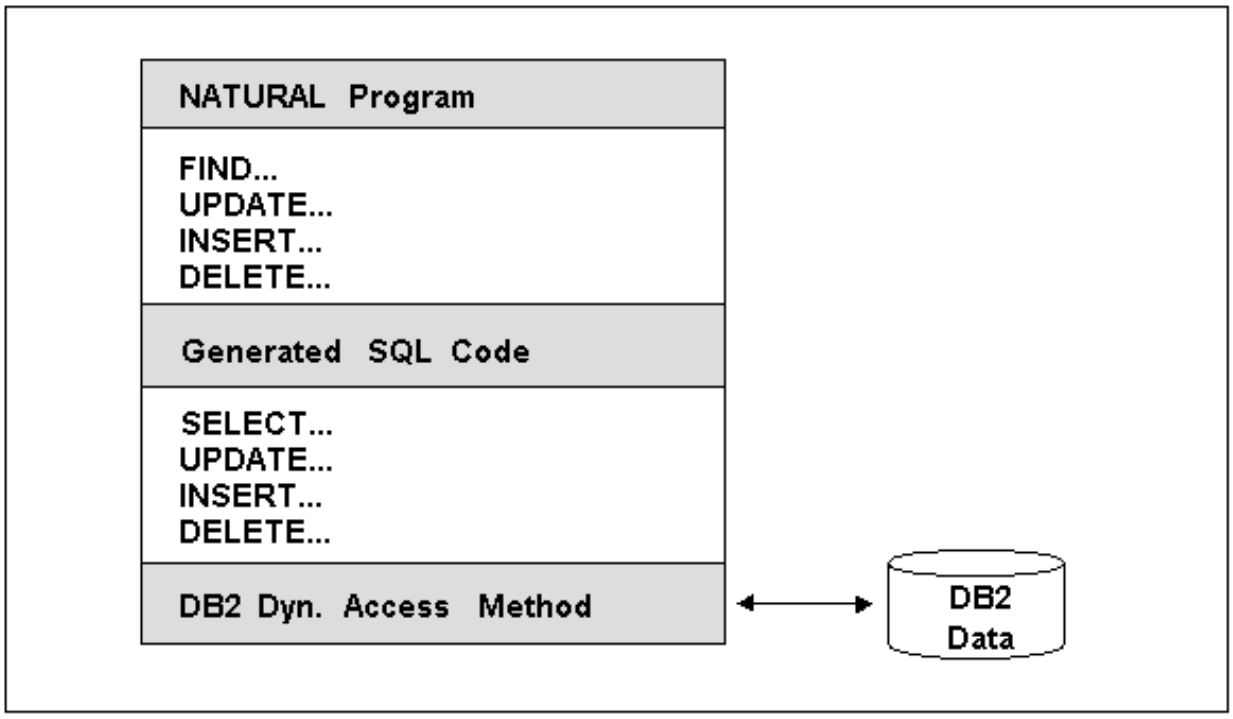
General Information

The SQL support of Natural combines the flexibility of dynamic SQL support with the high performance of static SQL support.

In contrast to static SQL support, the Natural dynamic SQL support does not require any special consideration with regard to the operation of the SQL interface. All SQL statements required to execute an application request are generated automatically and can be executed immediately with the Natural RUN command. Before executing a program, you can look at the generated SQL code, using the LISTSQL command.

Access to DB2 through Natural has the same form whether dynamic or static SQL support is used. Thus, with static SQL support, the same SQL statements in a Natural program can be executed in either dynamic or static mode. An SQL statement can be coded within a Natural program and, for testing purposes, it can be executed using dynamic SQL. If the test is successful, the SQL statement remains unchanged and static SQL for this program can be generated.

Thus, during application development, the programmer works in dynamic mode and all SQL statements are executed dynamically, whereas static SQL is only created for applications that have been transferred to production status.



Internal Handling of Dynamic Statements

Natural automatically provides for the preparation and execution of each SQL statement and handles the opening and closing of cursors used for scanning a table.

This section covers the following topics:

- NDBIOMO
- Statement Table
- Processing of SQL Statements Issued by Natural

NDBIOMO

As each dynamic execution of an SQL statement requires a statically defined DECLARE STATEMENT and DECLARE CURSOR statement, a special I/O module (NDBIOMO) is provided which contains a fixed number of these STATEMENTs and CURSORs. This number is specified during the generation of NDBIOMO.

Statement Table

If possible, an SQL statement is only prepared once and can then be executed several times if required. For this purpose, Natural internally maintains a table of all SQL statements that have been prepared and assigns each of these statements to a DECLAREd STATEMENT in NDBIOMO. In addition, this table maintains the cursors used by the SQL statements SELECT, FETCH, UPDATE (positioned), and DELETE (positioned).

Each SQL statement is uniquely identified by:

- the name of the Natural program that contains this SQL statement,
- the line number of the SQL statement in this program,
- the name of the Natural library, into which this program was stowed,
- the time stamp when this program was stowed.

Once a statement has been prepared, it can be executed several times with different variable values, using the dynamic SQL statement EXECUTE USING DESCRIPTOR or OPEN CURSOR USING DESCRIPTOR respectively.

When the full capacity of the statement table is reached, the entry for the next prepared statement overwrites the entry for a free statement whose latest execution is the least recent one.

When a new SELECT statement is requested, a free entry in the statement table with the corresponding cursor is assigned to it and all subsequent FETCH, UPDATE, and DELETE statements referring to this SELECT statement will use this cursor. Upon completion of the sequential scanning of the table, the cursor is released and free for another assignment. While the cursor is open, the entry in the statement table is marked as used and cannot be reused by another statement.

If the number of nested FIND (SELECT) statements reaches the number of entries available in the statement table, any further SQL statement is rejected at execution time and a Natural error message is returned.

The size of the statement table depends on the size specified for NDBIOMO. Since the statement table is contained in the DB2 buffer area, the DB2SIZE parameter may not be sufficient and may need to be increased.

Processing of SQL Statements Issued by Natural

The embedded SQL uses cursor logic to handle SELECT statements. The preparation and execution of a SELECT statement is done as follows:

1. The typical SELECT statement is prepared by a program flow which contains the following embedded SQL statements (note that *X* and *SQLOBJ* are SQL variables, not program labels):

   ```
   DECLARE SQLOBJ STATEMENT

   DECLARE X CURSOR FOR SQLOBJ

   INCLUDE SQLDA (copy SQL control block)

   Then, the following statement is moved into SQLSOURCE:

   SELECT PERSONNEL_ID, NAME, AGE FROM EMPLOYEES WHERE NAME IN (?, ?)    AND AGE BETWEEN ? AND ?
   ```

   (The question marks above are parameter markers which indicate where values are to be inserted at execution time.)

   ```
   PREPARE SQLOBJ FROM SQLSOURCE
   ```

2. Then, the SELECT statement is executed as follows:
   ```
   OPEN X USING DESCRIPTOR SQLDA
   FETCH X USING DESCRIPTOR SQLDA
   ```
   The descriptor SQLDA is used to indicate a variable list of program areas. When the OPEN statement is executed, it contains the address, length, and type of each value which replaces a parameter marker in the WHERE clause of the SELECT statement. When the FETCH statement is executed, it contains the address, length, and type of all program areas which receive fields read from the table.
   When the FETCH statement is executed for the first time, it sets the Natural system variable *NUMBER to a non-zero value if at least one record is found that meets the search criteria. Then, all records satisfying the search criteria are read by repeated execution of the FETCH statement.
   To help improve performance, especially when using distributed databases, the DB2-specific FOR FETCH ONLY clause can be used. FOR FETCH ONLY is generated and executed if rows are to be retrieved only; that is, if no updating is to take place.
3. Once all records have been read, the cursor is released by executing the following statement:
   ```
   CLOSE X
   ```


Preparing Natural Programs for Static Execution

This section covers the following topics:

- Basic Principles
- Generation Procedure - CMD CREATE Command
- Precompilation of the Generated Assembler Program
- Modification Procedure - CMD MODIFY Command
- BIND of the Precompiled DBRM

Basic Principles

Static SQL is generated in Natural batch mode for one or more Natural applications which can consist of one or more Natural object programs. The number of programs that can be modified for static execution in one run of the generation procedure is limited to 999.

During the generation procedure, the database access statements contained in the specified Natural objects are extracted, written to work files, and transformed into a temporary Assembler program. If no Natural program is found that contains SQL access or if any error occurs during static SQL generation, batch Natural terminates and condition code 40 is returned, which means that all further JCL steps should no longer be executed.

The Natural modules NDBCHNK and NDBSTAT must reside in a STEPLIB of the generation step. Both are loaded dynamically during the execution of the generation step.

The temporary Assembler program is written to a temporary file (the Natural work file CMWKF06) and precompiled. The size of the workfile is proportional to the maximum number of programs, the number of SQL statements and the number of variables used in the SQL statements. During the precompilation step, a Database Request Module (DBRM) is created, and after the precompilation step, the precompiler output is extracted from the Assembler program and written to the corresponding Natural objects, which means that the Natural objects are modified (prepared) for static execution. The temporary Assembler program is no longer used and deleted.

The Natural subprogram NDBDBRM can be used to check whether a Natural program contains SQL access and whether it has been modified for static execution.

A static DBRM is created by using either the sample job provided on the installation tape or an appropriate job created with the Create DBRM function.

Generation Procedure: CMD CREATE Command

To generate static SQL for Natural programs, LOGON to SYSDB2.

Since a new SYSDB2 library has been created when installing Natural for DB2, ensure that it contains all Predict interface programs necessary to run the static SQL generation. These programs are loaded into SYSDB2 at Predict installation time (see the relevant Predict documentation).

Then specify the CMD CREATE command and the Natural input necessary for the static SQL generation process; the CMD CREATE command has the following syntax:

CMD CREATE DBRM *static-name* **USING** *using-clause*

{*application-name,object-name,excluded-object***}**

:

:

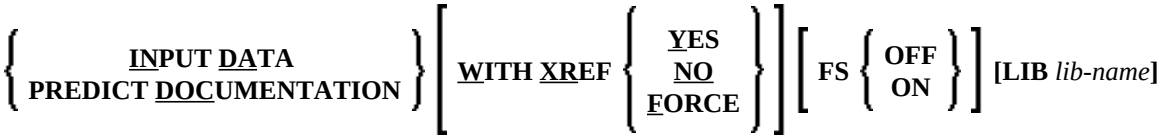
The generation procedure reads but does not modify the specified Natural objects. If one of the specified programs was not found or had no SQL access, return code 4 is returned at the end of the generation step.

Static Name

If the PREDICT DOCUMENTATION option is to be used, a corresponding Predict static SQL entry must be available and the *static-name* must correspond to the name of this entry. In addition, the *static-name* must correspond to the name of the DBRM to be created during precompilation. The *static-name* can be up to 8 characters long and must conform to Assembler naming conventions.

USING Clause

The *using-clause* specifies the Natural objects to be contained in the DBRM. These objects can either be specified explicitly as INPUT DATA in the JCL or obtained as PREDICT DOCUMENTATION from Predict.



If the parameters to be specified do not fit in one line, specify the command identifier (CMD) and the various parameters in separate lines and use both the input delimiter (as specified with the ID parameter; default is ",") and the continuation indicator (as specified with the CF parameter; default is "%") as shown in the following example:

Example:

```
CMD
CREATE,DBRM,static,USING,PREDICT,DOCUMENTATION,WITH,XREF,NO,%
LIB,library
```

Alternatively, you can also use abbreviations as shown in the following example:

Example:

```
CMD CRE DBRM static US IN DA W XR Y FS OFF LIB library
```

The sequence of the parameters USING, WITH, FS, and LIB is optional.

INPUT DATA

As input data, the applications and names of the Natural objects to be included in the DBRM must be specified in the subsequent lines of the job stream (*application-name,object-name*). A subset of these objects can also be excluded again (*excluded-objects*). Objects in libraries whose names begin with "SYS" can be used for static generation, too.

The applications and names of Natural objects must be separated by the input delimiter (as specified with the ID parameter; default is ","). If you wish to specify all objects whose names begin with a specific string of characters, use an *object-name* or *excluded-objects* name that ends with asterisk notation (*). To specify all objects in an application, use asterisk notation only.

Example:

```
LIB1,ABC*
LIB2,A*,AB*
LIB2,*
:
.
```

The specification of applications/objects must be terminated by a line that contains a period (.) only.

PREDICT DOCUMENTATION

Since Predict supports static SQL for DB2, you can also have Predict supply the input data for creating static SQL by using already existing PREDICT DOCUMENTATION.

WITH XREF Option

Since Predict Active References supports static SQL for DB2, the generated static DBRM can be documented in Predict and the documentation can be used and updated with Natural.

WITH XREF is the option which enables you to store cross-reference data for a static SQL entry in Predict each time a static DBRM is created (YES). You can instead specify that no cross-reference data are stored (NO) or that a check is made to determine whether a Predict static SQL entry for this static DBRM already exists (FORCE). If so, cross-reference data are stored; if not, the creation of the static DBRM is not allowed. For more detailed information on Predict Active References, refer to the relevant Predict documentation.

When WITH XREF (YES/FORCE) is specified, XREF data are written for both the Predict static SQL entry (if defined in Predict) and each generated static Natural program. However, static generation with WITH XREF (YES/FORCE) is possible only if the corresponding Natural programs have been cataloged with XREF ON.

WITH XREF FORCE only applies to the USING INPUT DATA option.

Note:
If you do not use Predict, the XREF option must be omitted or set to NO and the module NATXRF2 need not be linked to the Natural nucleus.

FS Option

If the FS (file server) option is set to ON, a second SELECT is generated for the Natural file server. ON is the default setting.

If the FS option is set to OFF, no second SELECT is generated, which results in less SQL statements being generated in your static DBRM and thus in a smaller DBRM.

LIB Option

With the LIB (library) option, a Predict library other than the default library (*SYSSTA*) can be specified to contain the Predict static SQL entry and XREF data. The name of the library can be up to eight characters long.

Precompilation of the Generated Assembler Program

In this step, the precompiler is invoked to precompile the generated temporary Assembler program. The precompiler output consists of the DBRM and a precompiled temporary Assembler program which contains all the database access statements transformed from SQL into Assembler statements.

Later, the DBRM serves as input for the BIND step and the Assembler program as input for the modification step.

The Modification Procedure: CMD MODIFY Command

The modification procedure modifies the Natural objects involved by writing precompiler information into the object and by marking the object header with the *static-name* as specified with the CMD CREATE command.

In addition, any existing copies of these objects in the Natural global buffer pool (if available) are deleted and XREF data are written to Predict (if specified during the generation procedure).

To perform the modification procedure, LOGON to SYSDB2 and specify the CMD MODIFY command which has the following syntax:

CMD MODIFY [XREF]

The input for the modify step is the precompiler output which must reside on a dataset defined as the Natural work file CMWKF01.

The output consists of precompiler information which is written to the corresponding Natural objects. In addition, a message is returned telling you whether it was the first time an object was modified for static execution ("modified") or whether it had been modified before ("re-modified").

If the XREF option is specified, the Natural work file CMWKF02 must be defined to contain the resulting list of cross-reference information concerning the statically generated SQL statements (see also Assembler/Natural Cross-References).

BIND of the Precompiled DBRM

It is recommended to run the BIND job **after** the MODIFY job.

This step performs the BIND against DB2. One or more DBRMs (as created by the precompiler) are processed to create a DB2 application plan. In addition to the static DBRMs created above, this application plan contains the dynamic DBRM NDBIOMO provided by Natural itself.

A DBRM can be bound into any number of application plans where it might be required. A plan is physically independent of the environment where the program is to be run. However, you can group your DBRMs logically into plans which are to be used for either batch or online processing, where the same DBRM can be part of both a batch plan and an online plan.

Unless you are using plan switching, only one plan can be executed per Natural session. Thus, you must ensure that the plan name specified in the BIND step is the same as the one used to execute Natural.

Assembler/Natural Cross-References

If you specify the XREF option of the MODIFY command, an output listing is created on the work file CMWKF02, which contains the DBRM name and the Assembler statement number of each statically generated SQL statement together with the corresponding Natural source code line number, program name, library name, database ID and file number.

```
-------------------------------------------------------------------------....
  DBRMNAME STMTNO      LINE NATPROG  NATLIB   DB  FNR COMMENT             ....
-------------------------------------------------------------------------....
  TESTDBRM 000627      0390 TESTPROG SAG      010 042 INSERT              ....
           000641      0430                          INSERT              ....
           000652      0510                          SELECT              ....
           000674      0570                          SELECT              ....
           000698      0570                          SELECT      2ND     ....
           000728      0650                          UPD/DEL             ....
           000738      0650                          UPD/DEL     2ND     ....
           000751      0700                          SELECT              ....
           000775      0700                          SELECT      2ND     ....
```

Column	Explanation
DBRMNAME	Name of the DBRM which contains the static SQL statement.
STMTNO	Assembler statement number of the static SQL statement.
LINE	Corresponding Natural source code line number.
NATPROG	Name of the Natural program that contains the static SQL statement.
NATLIB	Name of the Natural library that contains the Natural program.
DB / FNR	Natural database ID and file number.
COMMENT	Type of SQL statement, where "2ND" indicates that the corresponding statement is used for a reselection; see also the Concept of the File Server.

Execution of Natural in Static Mode

To be able to execute Natural in static mode, all users of Natural must have the DB2 EXECUTE PLAN/PACKAGE privilege for the plan created in the BIND step.

To execute static SQL start Natural and execute the corresponding Natural program. Internally, the Natural runtime interface evaluates the precompiler data written to the Natural object and then performs the static accesses.

To the user there is no difference between dynamic and static execution.

Static SQL with Natural Security

Static generation can be disallowed with Natural Security by:

- restricting access to library SYSDB2,
- disallowing the module CMD,
- restricting access to the libraries that contain the relevant Natural objects,
- disallowing one of the commands CATALOG, or STOW for a library that contains relevant Natural objects.

If a library is defined in Natural Security and the DBID and FNR of this library are different from the default specifications, the static generation procedure automatically switches to the DBID and FNR specifications defined in Natural Security.

Mixed Dynamic/Static Mode

It is possible to operate Natural in a mixed static and dynamic mode where for some programs static SQL is generated and for some not.

The mode in which a program is run is determined by the Natural object program itself. If a static DBRM is referenced in the executing program, all statements in this program are executed in static mode.

Note:
Natural programs which return a runtime error do not automatically execute in dynamic mode. Instead, either the error must be corrected or, as a temporary solution, the Natural program must be recataloged to be able to execute in dynamic mode.

Within the same Natural session, static and dynamic programs can be mixed without any further specifications. The decision which mode to use is made by each individual Natural program.

Messages and Codes

This section lists the error messages that may be issued during static generation.

STAT9001 Object buffer allocation failed. RC = *return code*

Program NDBCHNK has been invoked to allocate space for Natural object load, but the allocation has failed; retry or increase the free storage pool.

STAT9002 Write on object area failed. RC = *return code*

Program NDBCHNK has been invoked to write a Natural object row into the appropriate buffer, but the write has failed; this is probably a NDBCHNK program error.

STAT9003 Statement entry retrieve error. RC = *return code*

Program NDBSTAT has been invoked to retrieve next DB2 statement information from the Natural object loaded in main storage, but the retrieval has failed (RC was neither 0 (OK) nor 4 (EOP)); the probable cause is a Natural object inconsistency.

STAT9004 Unsupported Adabas command: *command*

Program NDBSTAT has been invoked to retrieve next DB2 statement information from the Natural object loaded in main storage, but the Adabas command code returned was invalid; the probable cause is a Natural object inconsistency.

STAT9005 Freemain failed. RC = *return code*

Program NDBCHNK has been invoked to free the area allocated for Natural object load, but the release has failed; this is probably a program error.

STAT9006 Call for timestamp of program failed. RC = *return code*

Program NDBSTAT has been invoked to know the time stamp associated to the loaded Natural object, but the call has failed; this is probably a program error.

STAT9007 A-List item retrieve failed. RC = *return code*

Program NDBSTAT has been invoked to retrieve the next compilation A-list element, but the retrieval has failed (RC was neither 0 (OK) nor 20 (EOL)); the probable cause is a Natural object inconsistency.

STAT9009 Invalid database field format: *format*

Program NDBSTAT has been invoked to retrieve the next compilation A-list element, but the DB2 format code returned is invalid; the probable cause is a Natural object inconsistency.

STAT9014 Warning, may indicate a problem: second select table reset.

The table for a second selection logs the statement number of all second SELECT statements. The table is reset if there are more than 100 entries, which means with many nested program loops. If the table is reset, no second UPDATE or DELETE statements are generated.

STAT9016 Versions of NDBSTAT and SQLGEN Natural programs do not match.

The versions of the Natural programs used for the static generation (library SYSSQL) must be the same as one of the dynamically loaded Assembler program NDBSTAT.

STAT9017 *address* **of program** *program* **in library** *library* **not found.**

A Natural object address was not found and the object cannot be modified. Either the object was not found or the address was wrong.

STAT9019 * Warning: Natural terminates abnormally, run may continue. ***

Warning: Natural terminates abnormally with RC=4. A Natural member was explicitly entered which does not exist or does not have SQL access. The static generation can continue.

STAT9020 Start run of SQLGEN for DBRM *dbrm***.**

STAT9021 Start merging temporary datasets.

STAT9022 Precompile input *input* **written to temporary dataset.**

The temporary assembler program for the precompiler input was written to a temporary dataset (Natural work file 6).

STAT9023 * END OF DATA ***

STAT9024 No program with SQL access found.

None of the programs processed by the CMD command accessed an SQL system.

STAT9025 Program *program* **in library** *library* **not found.**

STAT9026 DB access module names *module* **and** *module* **do not match.**

The module name specified with the CMD CREATE command must be the same as the name of the DBRM specified in the DBRMLIB job card of the precompilation step.

STAT9027 Error *error* **purging** *program***,** *library* **in buffer pool. Run continues.**

STAT9028 Number of programs to be generated exceeds 999.

The number of programs to be generated statically into one DBRM exceeds the maximum number of 999.

STAT9029 Limit of *limit* **NULL indicators per SQL statement exceeded.**

The maximum number of 1500 NULL indicators per SQL statement has been exceeded.

STAT9030 Number of variables to be generated exceeds 9999.

The number of variables to be generated statically for one program exceeds the maximum number of 9999.

STAT9031 XREF option "NO" and Predict DDA default "YES" do not match.

The Predict DDA default setting for static SQL XREF is set to "YES" but the XREF option in the CMD command is set to "NO".

STAT9032 XREF option "FORCE" but no Predict documentation found.

With the XREF option FORCE, the static generation continues and writes XREF data only if Predict documentation exists for a given DBRM. If there is no Predict documentation available, static generation is not performed.

STAT9033 No XREF data exist for member *member***.**

Either the Natural program which is to be statically generated cannot be cataloged with XREF=ON or the XREF data are not on the used Predict file.

STAT9034 XREF option "YES" or "NO" but Predict DDA default "FORCE".

The Predict DDA default setting for static SQL XREF is set to "FORCE", but the XREF option in the CMD command is set to "NO" or "YES.

STAT9036 Given DBRM library not defined as 3GL Predict application.

The library for the DBRM entered with the LIB option is not defined as 3GL application in Predict. Check the library name in Predict which contains the DBRM.

STAT9039 Library name must not be blank.

STAT9040 CAT or STOW not allowed for library *library***.**

The commands CAT or STOW are not allowed in your security environment. However, the CAT or STOW privilege is needed for static generation.

STAT9041 Natural Security restriction. Message code: *message code*

STAT9050 No Predict documentation for specified DBRM found.

No documentation was found in Predict for the DBRM specified with the CMD command. Either the DBRM is not documented in the used Predict file or a wrong DBRM name has been specified.

STAT9062 No Predict installed or SM level less than SM4.

STAT9063 XREF interface not linked. XREF option reset, run continues.

STAT9064 XREF option not set. Predict DDA default *default* **taken.**

The Predict DDA default setting for static SQL XREF is read, because no XREF option is specified in the CMD command and the XREF interface and Predict are installed.

STAT9065 DBRM name must start with an uppercase character.

STAT9066 No Predict installed or SM level less than SM4, run continues.

STAT9072 DBRM name must not be blank.

STAT9073 Invalid syntax for *parameter/option* **specified.**

STAT9092 Error occurred. XREF data for DBRM will be deleted.

STAT9093 Error *error* **occurred in program** *program* **on line** *line***.**

STAT9094 Return code *return code* **on call of** *program***.**

STAT9095 Error in *parameter* **parameter on call of** *program***.**

Application Plan Switching in Static SQL

When using application plan switching, you can switch to a different application plan within the same Natural session.

If a second application plan is to be used, this can be specified by executing the Natural program NATPLAN. NATPLAN is contained in the library SYSDB2 and can be invoked either from within a Natural program or dynamically by entering the command NATPLAN in the NEXT line. The only input value required for NATPLAN is an eight-character plan name. If no plan name is specified, you are prompted by the system to do so.

Before executing NATPLAN, ensure that any open DB2 recovery units are closed.

Since the NATPLAN program is also provided in source form, user-written plan switching programs can be created using similar logic.

The actual switch from one plan to another differs in the various environments supported. The feature is available under Com-plete, CICS, and IMS/TM MPP. When using the CAF interface, it is also available in TSO and batch environments.

In some of these environments, a transaction ID or code must be specified instead of a plan name.

This section covers the following topics:

- Plan Switching under CICS
- Plan Switching under Com-plete
- Plan Switching under IMS/TM
- Plan Switching under TSO and in Batch Mode

Plan Switching under CICS

Under CICS, you have the option of using either plan switching by transaction ID (default) or dynamic plan selection exit routines. Thus, by setting the field "#SWITCH-BY- TRANSACTION-ID" in the NATPLAN program to either "TRUE" or "FALSE", either the subroutine CMTRNSET or the module NDBPLNA is invoked.

See more information on activating plan switching under CICS.

Plan Switching by Transaction ID

If "#SWITCH-BY-TRANSACTION-ID" is set to "TRUE", the subroutine CMTRNSET is invoked, which changes the current pseudo-conversational transaction ID to the one you want to switch to.

CMTRNSET is documented in the library SYSEXTP, which is invoked with the SYSEXTP system command. After calling CMTRNSET, you have to perform a terminal I/O to ensure that a new CICS transaction is used.

Using plan switching by transaction ID enables you to use existing CICS entry threads for your Natural transactions, where each transaction ID can have its own entry thread assigned.

Plan Switching by CICS/DB2 Exit Routine

If "#SWITCH-BY-TRANSACTION-ID" is set to "FALSE", the NDBPLNA module is invoked to supply a specified CICS/DB2 exit routine with the plan name to be used. Thus, for a Natural application to perform plan switching by a CICS/DB2 exit routine, the NATPLAN program must be invoked before the first DB2 access. For additional information on CICS/DB2 exit routines, refer to the relevant IBM literature.

NDBPLNA writes a CICS temporary storage record containing the plan name that was supplied to NATPLAN. The name of the temporary storage queue is PLANXXXX, where XXXX is the CICS terminal identifier.

When running in a CICSplex environment, the CICS temporary storage queue PLANXXXX containing the plan name must be defined with TYPE=SHARED or TYPE=REMOTE in a CICS TST.

For each new DB2 unit of recovery, the appropriate plan selection exit routine is automatically invoked. This exit routine reads the temporary storage record and uses the contained plan name for plan selection.

When no temporary storage record exists for the Natural session, a default plan name, contained in the plan exit, can be used. If no plan name is specified by the exit, the name of the plan used is the same as the name of the static program (DBRM) issuing the SQL call. If no such plan name exists, an SQL error results.

Plan Switching under Com-plete

In Com-plete environments, plan switching is accomplished using the Call Attachment Facility (CAF) to release the thread in use and attach another one that has a different plan name.

Once the DB2 connection has been established, the same plan name continues to be used until the plan is explicitly changed using the call attachment language interface (DSNALI). For additional information on the CAF interface, refer to the relevant IBM literature.

Under Com-plete, the NATPLAN program first issues an END TRANSACTION statement and then invokes an Assembler module named NDBPLAN by using DB2SERV.

NDBPLAN performs the actual switching. It issues a CLOSE request to DSNALI to terminate the DB2 connection (if one exists). It then issues an OPEN request to re-establish the DB2 connection and to allocate the resources needed to execute the specified plan.

If NATPLAN has not been executed before the first SQL call, the default plan used is the one defined in the Com-plete startup parameters. Once a plan has been changed using NDBPLAN, it remains scheduled until another plan is scheduled by NDBPLAN or until the end of the Natural session.

Plan Switching under IMS/TM

In an IMS MPP environment, the switch is accomplished using direct or deferred message switching. As a different application plan is associated with each IMS application program, message switching from one transaction code to another automatically switches the application plan being used.

Since Natural applications can perform direct or deferred message switches by calling the appropriate supplied routines, use of the NATPLAN program for plan switching is optional.

NATPLAN calls the Assembler routine CMDEFSW, which sets the new transaction code to be used with the next following terminal I/O.

Plan Switching under TSO and in Batch Mode

In the TSO and batch environments, plan switching is accomplished using the Call Attachment Facility (CAF) to release the thread in use and attach another one that has a different plan name. Using CAF, plan selection is either implicit or explicit.

If no DB2 connection has been made before the first SQL call, a plan name is implicitly selected by DB2. If so, the plan name used is the same as the name of the program (DBRM) issuing the SQL call.

Once the DB2 connection has been established, the same plan name continues to be used until the plan is explicitly changed using the call attachment language interface (DSNALI). For additional information on the CAF interface, refer to the relevant IBM literature.

Under TSO and in batch mode, the NATPLAN program first issues an END TRANSACTION statement and then invokes an Assembler module named NDBPLAN by using DB2SERV.

Note:
Modify the NATPLAN program by setting the #SSM field to the current DB2 subsystem name; the default name is DB2.

NDBPLAN performs the actual switching. It issues a CLOSE request to DSNALI to terminate a possible DB2 connection. It then issues an OPEN request to re-establish the DB2 connection and to allocate the resources needed to execute the specified plan.

If NATPLAN has not been executed before the first SQL call, plan selection is done implicitly by DSNALI. If so, the plan name used is the same as the name of the program issuing the SQL call. The subsystem ID used is the one specified during the DB2 installation. If no such plan name or subsystem ID exists, a Natural error message is returned.

If a static DBRM is issuing the SQL call, a plan name must exist with the same name as the one of the static DBRM.

If dynamic SQL is being used, a plan name of NDBIOMO must exist which must contain a DBRM called NDBIOMO, too. NDBIOMO is the default plan name.

Note:
To avoid any confusion concerning the chosen plan name and/or subsystem ID, always call NATPLAN before the first SQL call.

